Course Outline

- **Day1** ✓ DONE
  - Introduction to Java
  - Basic Java Concepts
- **Day2** ✓ DONE
  - Data Types & Operators
  - using Arrays & Strings
  - Controlling Program Flow
  - Modifiers and Access Specifiers
- **Day3** ✓ DONE
  - Simple GUI
  - Essential Java Classes
  - Java Exception
- **Day4** ✓ DONE
  - Interfaces
  - Multi-Threading
- **Day5**
  - Inner class
  - Event Handling

Java Programming
Inner Classes

Presented By
Dr. Eman Hesham, DBA.

Java™ Education
and Technology Services

Invest In Yourself ,
Develop Your Career

Inner Classes

- The Java programming language allows you to define a class within another class.
  - Such a class is called a ***nested class [Inner Class]***.

```
class OuterClass
 {
        ...
        class InnerClass
        {
                ...
        }
}
```

Types of Inner Classes

- There are broadly four types of inner classes:
    1. Normal Member Inner Class
    2. Static Member Inner Class
    3. Local Inner Class (inside method body)
    4. Local Anonymous Inner Class

```java
public class OuterClass{
  private int x ;
  public void myMethod(){
       MyInnerClass m = new MyInnerClass();
       m.aMethod();
       ..
  }
  public class MyInnerClass{
       public void aMethod(){
            //you can access private members of the outer class here
            x = 3 ;
       }
  }
}
```

1. Normal Member Inner Class

- In order to create an object of the inner class you need to use an object of the outer class.
- The following line of code could have been written inside the method of the enclosing class:

```
MyInnerClass m = this.new MyInnerClass();
```

- The following line of code is used to create an object of the inner class outside of the enclosing class:
-

```
OuterClass obj = new OuterClass() ;
OuterClass.MyInnerClass m = obj.new MyInnerClass();
```

1. Normal Member Inner Class

- An inner class can extend any class and/or implement any interface.
- An inner class can assume any accessibility level:
  - private,protected, or public.
- An inner class can have an inner class inside it.
- When you compile the java file, two class files will be produced:
  - MyClass.class
  - MyClass$MyInnerClass.class
- The inner class has an implicit reference to the outer class.

1. Normal Member Inner Class

```
public class MyClass{
   private int x ;
   public void myMethod(){
       MyInnerClass m = new MyInnerClass();
       m.aMethod();
   }
   class MyInnerClass{
       int x ;
       public void aMethod(){
           x = 10 ;  //x of the inner class
           MyClass.this.x = 25 ;  // x of the outer class
       }
   }
}
```

The inner class has an implicit reference to the outer class

2. Static Inner Class

- You know, The normal inner class has implicitly a reference to the outer class that created it.
    - If you don’t need a connection between them, then you can make the **inner class static**.
- **A static inner class** means:
    - You don’t need an outer-class object in order to create an object of a static inner class.
    - You can’t access an outer-class object from an object of a static inner class.

2. Static Inner Class

- Static Inner Class:
  - is among the static members of the outer class.
  - When you create an object of static inner class, you don’t need to use an object of the outer class (remember: it’s static!).
  - Since it is static, such inner class will only be able to access the static members of the outer class.

2. Static Inner Class (Example)

```
public class OuterClass{

   int x ;
   static int y;

   public static class InnerClass{
        public void aMethod(){
        y = 10;                 // OK
        x = 33;                 // wrong
    }
   }
}
```

```
InnerClass ic= OuterClass.new InnerClass();
```

3. Local Inner Class

```
public class MyClass {
   private int x ;
   public void myMethod(final String str, final int a){
        final int b;
        class MyLocalInnerClass{
            public void aMethod(){
                //you can access private members of the outer class
                //and you can access final local variables of the method
            }
        }
       MyLocalInnerClass myObj = new MyLocalInnerClass();
    }
}
```

3. Local Inner Class

- The object of the local inner class can only be created below the definition of the local inner class (within the same method).
- The local inner class can access the member variables of the outer class.
- It can also access the local variables of the enclosing method if they are declared final.

4. Anonymous Inner Class

```
public class MyClass
{
   int x ;
   public void init()
   {
     Thread th = new Thread(new Runnable()
                                {
                                    public void run()
                                    {
                                         ..
                                    }
                                });
     th.start();
   }
}
```

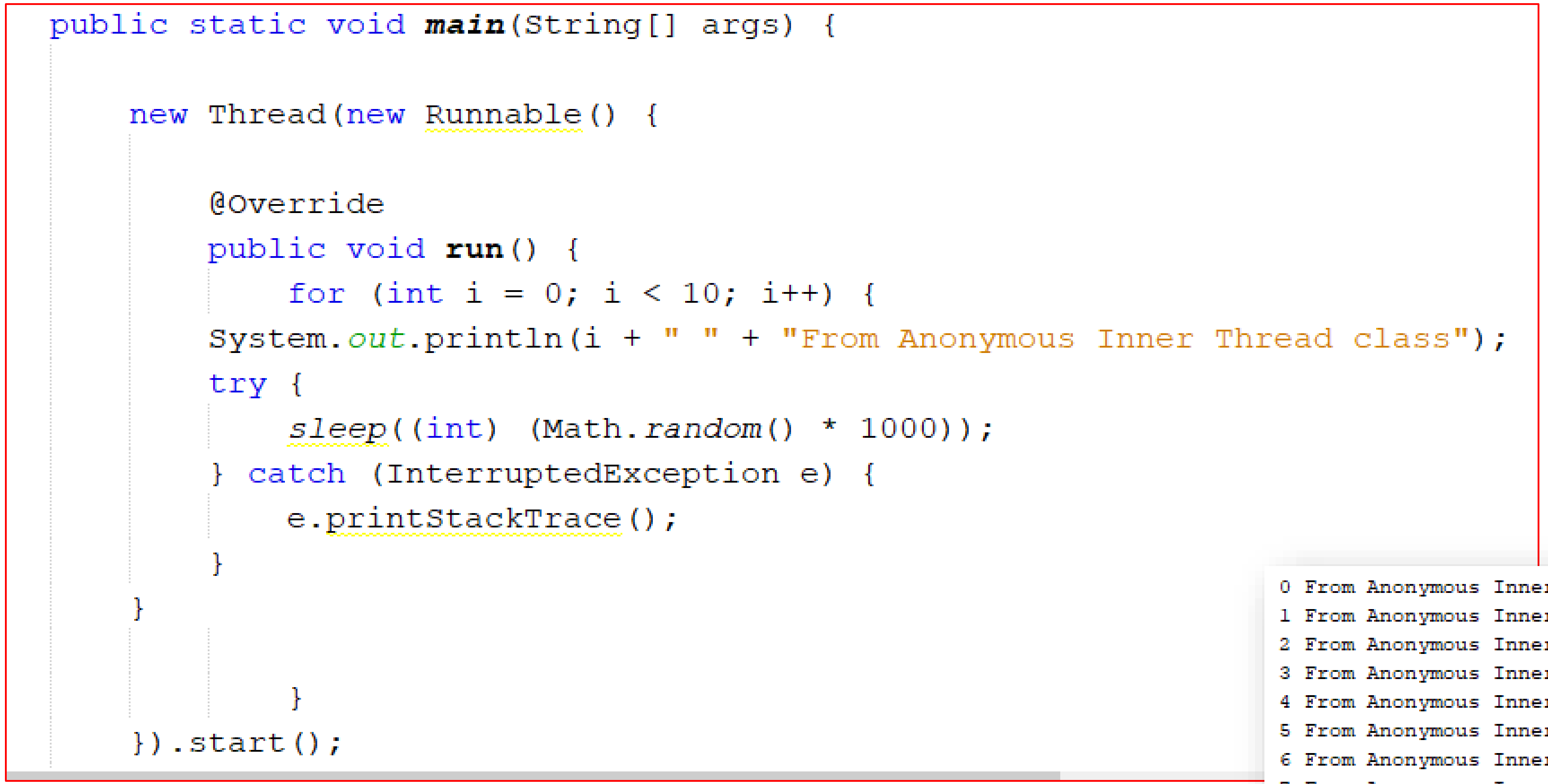
4. Anonymous Inner Class

- The whole point of using an anonymous inner class is to implement an interface or extend a class and then override one or more methods.

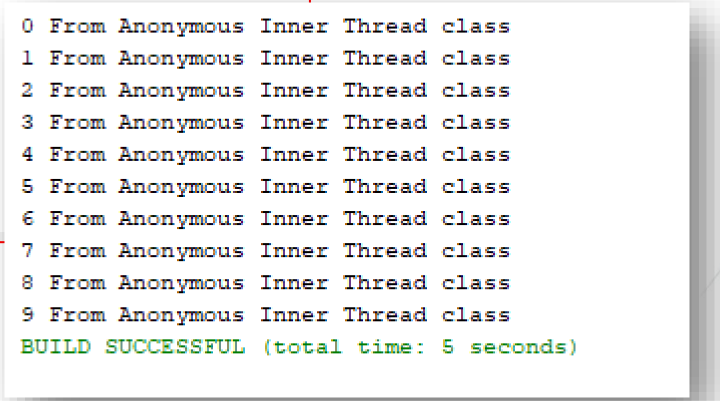
- Of course, it does not make sense to define new methods in anonymous inner class; how will you invoke it?

- When you compile the java file, two class files will be produced:
  - MyClass.class
  - MyClass$1.class

Example:



```
public static void main(String[] args) {

    new Thread(new Runnable() {

        @Override
        public void run() {
            for (int i = 0; i < 10; i++) {
        System.out.println(i + " " + "From Anonymous Inner Thread class");
        try {
            sleep((int) (Math.random() * 1000));
        } catch (InterruptedException e) {
            e.printStackTrace();
        }
    }

            }
    }).start();
```



```
0 From Anonymous Inner Thread class
1 From Anonymous Inner Thread class
2 From Anonymous Inner Thread class
3 From Anonymous Inner Thread class
4 From Anonymous Inner Thread class
5 From Anonymous Inner Thread class
6 From Anonymous Inner Thread class
7 From Anonymous Inner Thread class
8 From Anonymous Inner Thread class
9 From Anonymous Inner Thread class
BUILD SUCCESSFUL (total time: 5 seconds)
```

Java Programming
Event Handling

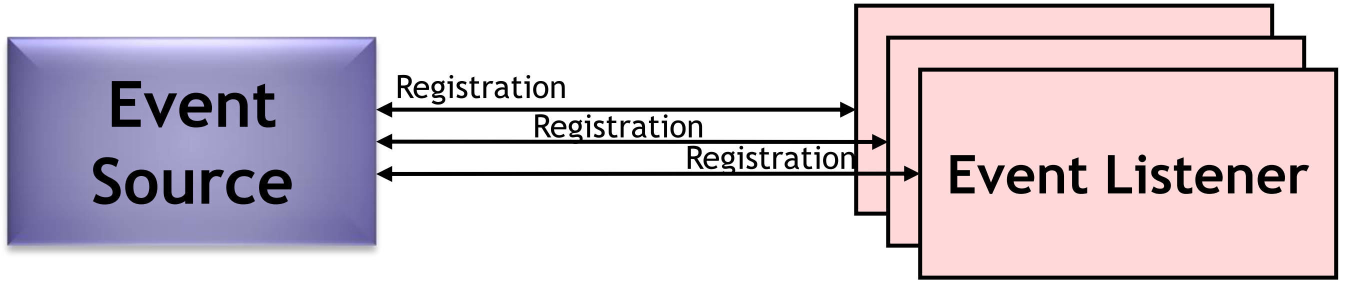
Java™ Education
and Technology Services

Invest In Yourself ,
Develop Your Career

Event Handling

- Event Delegation Model was introduced to Java since JDK 1.1
- This model realizes the event handling process as two roles:
  - Event Source
  - Event Listener.
- The Source:
  - is the object that fired the event,
- The Listener:
  - is the object that has the code to execute when notified that the event has been fired.

Event Handling



- An Event Source may have one or more Event Listeners.
- The advantage of this model is:
  - The Event Object is only forwarded to the listeners that have registered with the source.

Event Handling

- When working with GUI, the source is usually one of the GUI Components (e.g. Button, Checkbox, …etc).
- The following piece of code is a simplified example of the process:

```
JButton b = new JButton(“Ok”); //Constructing a Component (Source)
MyListener myL = new MyListener(); //Constructing a Listener
b.addActionListener(myL); //Registering Listener with Source
```

Event Handling

- Let’s look at the signature of the registration method:

```
void addActionListener(ActionListener l)
```

- In order for a listener to register with a source for a certain event,
  - it has to implement the proper interface that corresponds to the designated event, which will enforce a certain method to exist in the listener.

```
class MyListener implements ActionListener {

   public void actionPerformed(ActionEvent e) {
        // handle the event here
        // (i.e. what you want to do
        // when the Ok button is clicked)
   }
}
```

Event Handling

- Let’s look at the signature of the registration method:

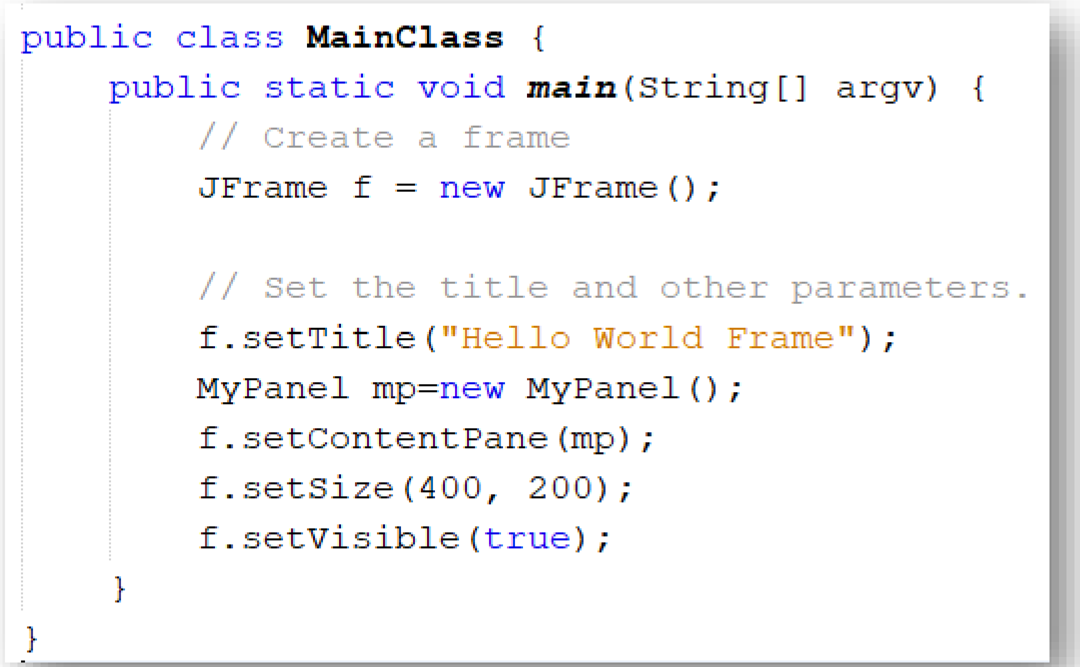
```
void addActionListener(ActionListener l)
```

- In order for a listener to register with a source for a certain event,
  - it has to implement the proper interface that corresponds to the designated event, which will enforce a certain method to exist in the listener.

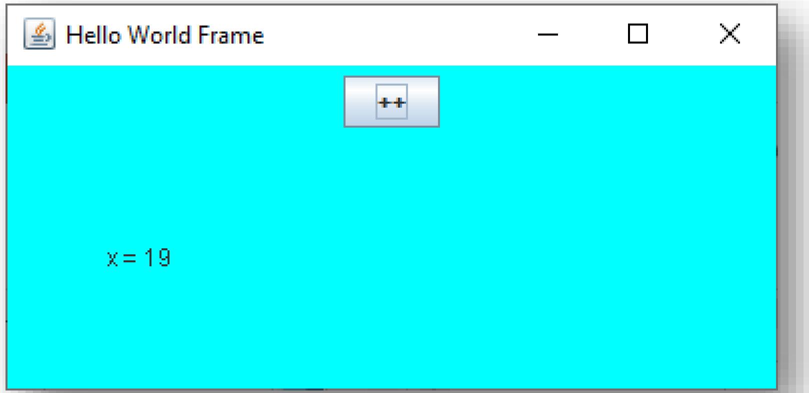
```
class MyListener implements ActionListener {

   public void actionPerformed(ActionEvent e) {
        // handle the event here
        // (i.e. what you want to do
        // when the Ok button is clicked)
   }
}
```

Example

- Create a frame that has a button to increment the counter value.



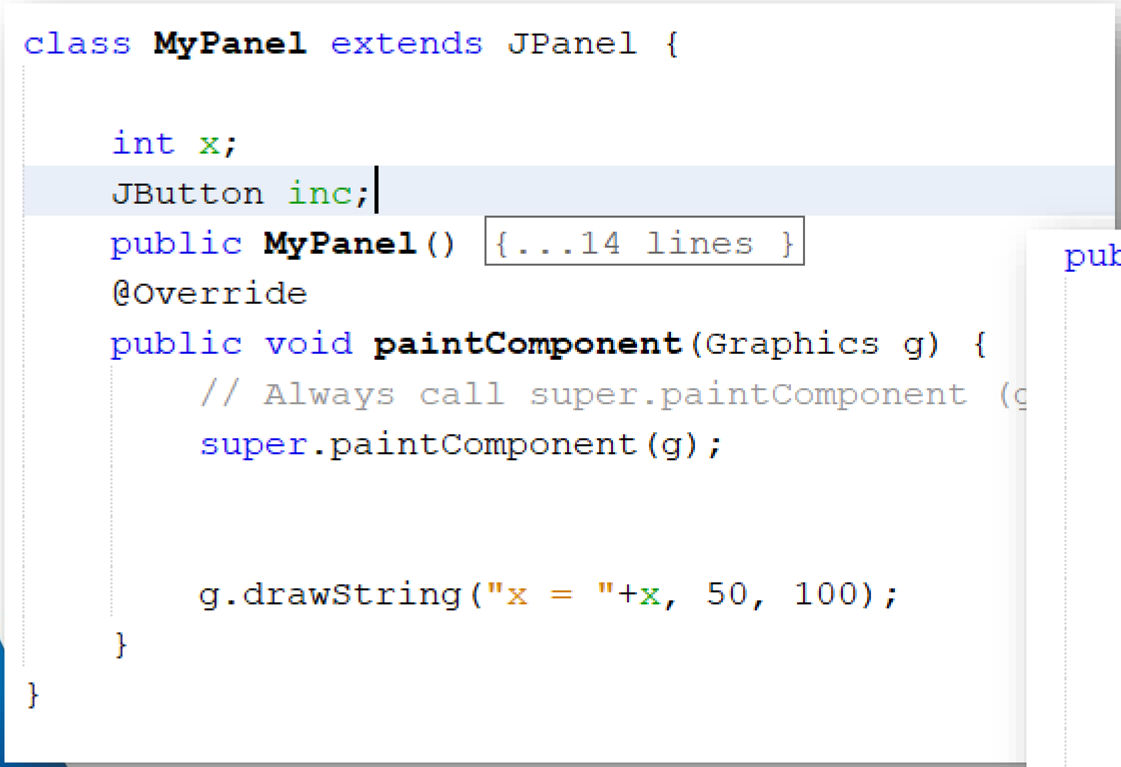
```
public class MainClass {
    public static void main(String[] argv) {
        // Create a frame
        JFrame f = new JFrame();

        // Set the title and other parameters.
        f.setTitle("Hello World Frame");
        MyPanel mp=new MyPanel();
        f.setContentPane(mp);
        f.setSize(400, 200);
        f.setVisible(true);
    }
}
```



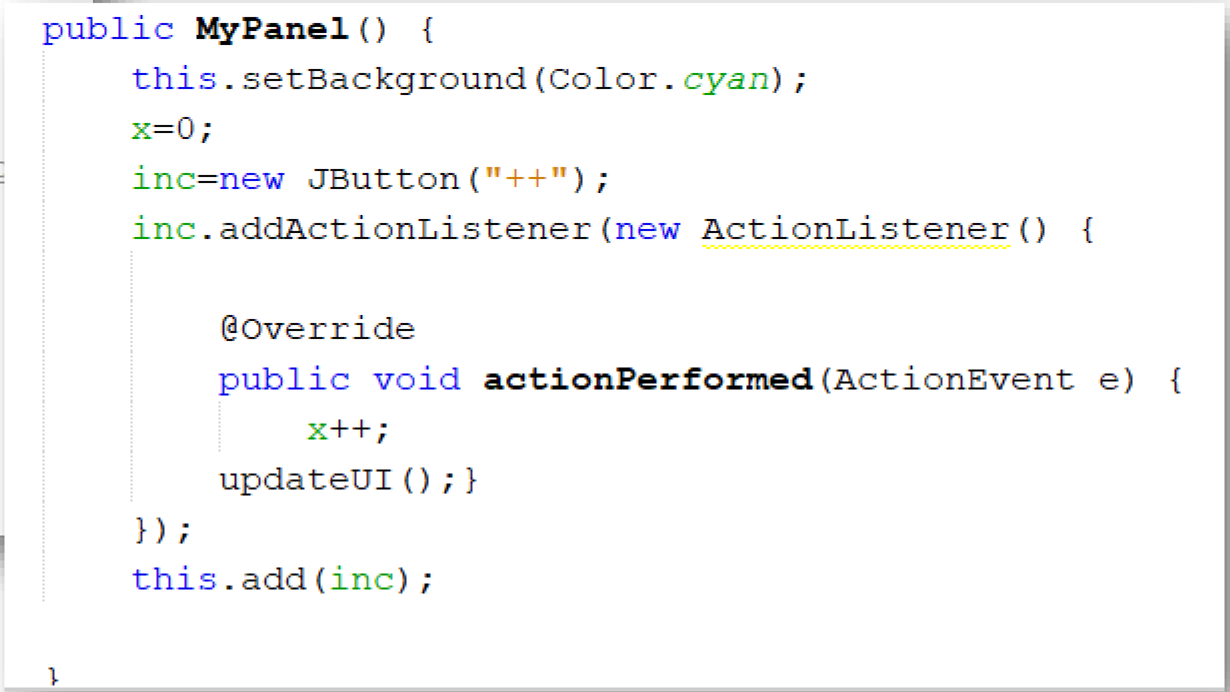
```
x = 19
```

Example

- Create a frame that has a button to increment the counter value.



```
class MyPanel extends JPanel {

    int x;
    JButton inc;
    public MyPanel() {...14 lines }
    @Override
    public void paintComponent(Graphics g) {
        // Always call super.paintComponent (g
        super.paintComponent(g);


        g.drawString("x = "+x, 50, 100);
    }
}
```



```
public MyPanel() {
    this.setBackground(Color.cyan);
    x=0;
    inc=new JButton("++");
    inc.addActionListener(new ActionListener() {

        @Override
        public void actionPerformed(ActionEvent e) {
            x++;
        updateUI();}
    });
    this.add(inc);

}
```

Java™ Education and Technology Services

Events Classes and Listener Interfaces

Event	Listener Interface(s)	Method(s)
ActionEvent	ActionListener	actionPerformed (ActionEvent e)
AdjustmentEvent	AdjustmentListener	adjustmentValueChanged (AdjustmentEvent e)
ComponentEvent	ComponentListener	componentHidden (ComponentEvent e) componentShown (ComponentEvent e) componentMoved (ComponentEvent e) componentResized (ComponentEvent e)
ItemEvent	ItemListener	itemStateChanged (ItemEvent e)
TextEvent	TextListener	textValueChanged (TextEvent e)
ContainerEvent	ContainerListener	componentAdded (ComponentEvent e) componentRemoved (ComponentEvent e)

Events Classes and Listener Interfaces

Event	Listener Interface(s)	Method(s)
FocusEvent	FocusListener	focusGained (FocusEvent e) focusLost (FocusEvent e)
WindowEvent	WindowListener	windowClosed (WindowEvent e) windowClosing (WindowEvent e) windowOpened (WindowEvent e) windowActivated (WindowEvent e) windowDeactivated (WindowEvent e) windowIconified (WindowEvent e) windowDeiconfied (WindowEvent e)

Java™ Education and Technology Services

Events Classes and Listener Interfaces

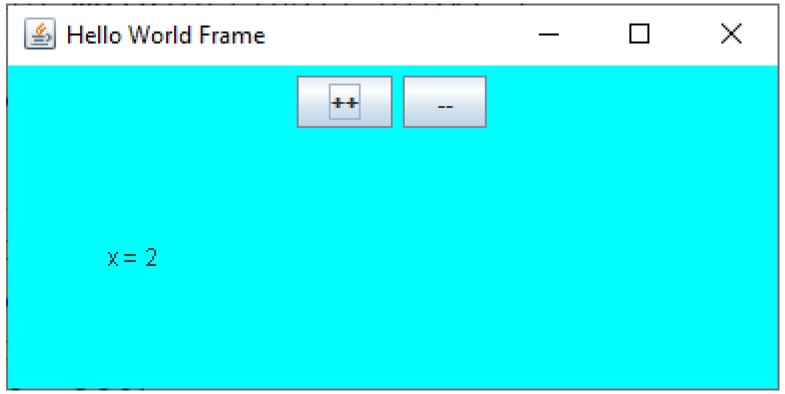
Event	Listener Interface(s)	Method(s)
KeyEvent	KeyListener	keyPressed (KeyEvent e) keyReleased (KeyEvent e) keyTyped (KeyEvent e)
MouseEvent	MouseListener	mousePressed (MouseEvent e) mouseReleased (MouseEvent e) mouseClicked (MouseEvent e) mouseEntered (MouseEvent e) mouseExited (MouseEvent e)
	MouseMotionListener	mouseMoved (MouseEvent e) mouseDragged (MouseEvent e)
MouseWheel Event	MouseWheelListener	mouseWheelMoved (MouseWheelEvent e)

Java™ Education and Technology Services

Lab Exercise

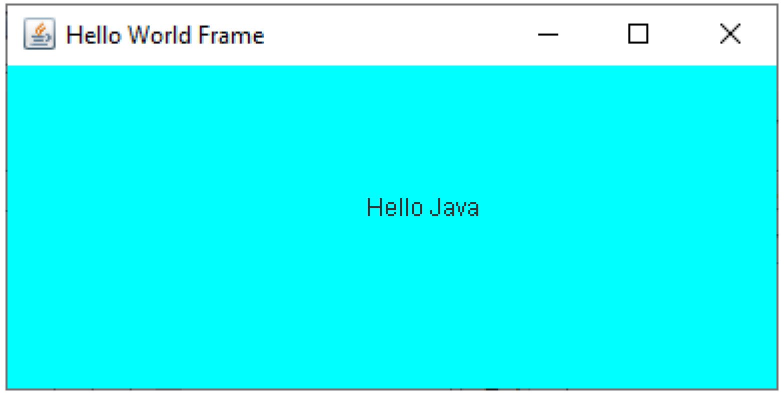
Lab Exercise 1

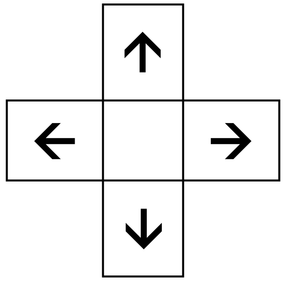
- Create a frame that has two buttons one to increment the counter value and one to decrement this value.



Lab Exercise 2

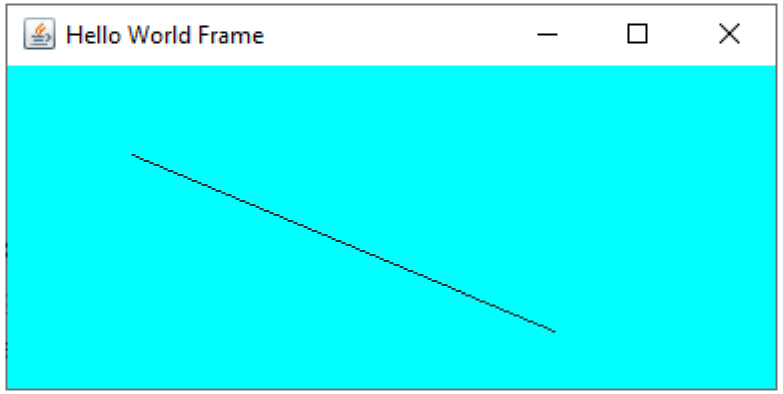
- Create a frame that displays string which user can move it using arrow keys.





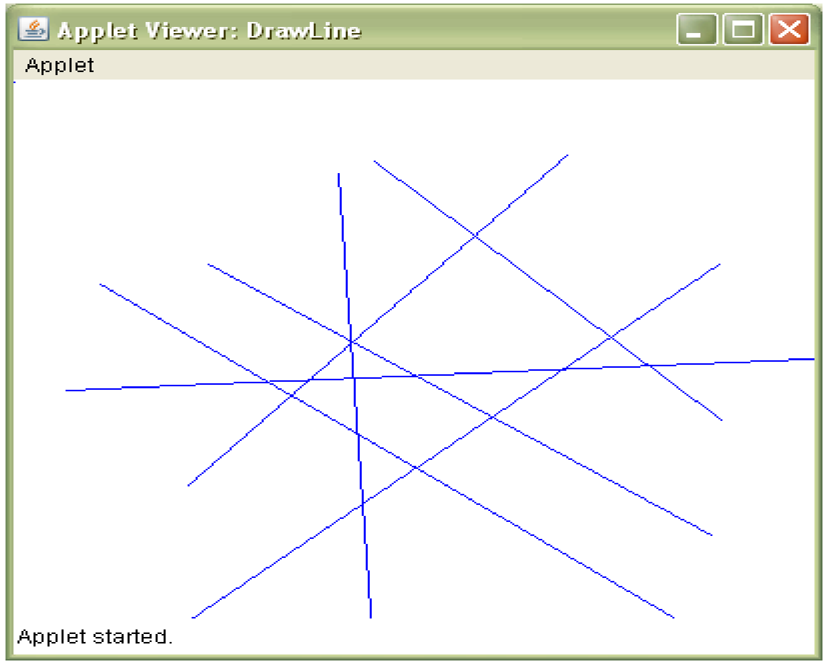
Lab Exercise 3

- Create a frame that allows the user to draw one line by dragging the mouse on the frame.



Lab Exercise 4

- Modify the previous exercise to allow the user to draw multiple lines on the frame.
- Store the drawn lines in an array to be able to redraw them again when the panel is repainted.



Java™ Education and Technology Services

Course Outline

- **Day1** (DONE)
  - Introduction to Java
  - Basic Java Concepts
- **Day2** (DONE)
  - Data Types & Operators
  - using Arrays & Strings
  - Controlling Program Flow
  - Modifiers and Access Specifiers
- **Day3** (DONE)
  - Simple GUI
  - Essential Java Classes
  - Java Exception
- **Day4** (DONE)
  - Interfaces
  - Multi-Threading
- **Day5** (DONE)
  - Inner class
  - Event Handling